| Materia: | Programación II | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nivel: | 2º Cuatrimestre | | | | | | | | | | | |
| Tipo de Examen: | Segundo Parcial | | | | | | | | | | | |
| Apellido**(1)**: |  | | | | Fecha: | | | | 29 nov 2024 | | | |
| Nombre/s**(1)**: |  | | | | Docente a cargo**(2)**: | | | | Benitez Silvio Daniel | | | |
| División**(1)**: |  | | | | Nota**(2)**: | | | |  | | | |
| DNI**(1)**: |  | | | | Firma**(2)**: | | | |  | | | |

**(1)** Campos a completar solo por el estudiante en caso de imprimir este enunciado en papel.

**(2)** Campos a completar solo por el docente en caso de imprimir este enunciado en papel.

**Enunciado:**

Una agencia de turismo necesita un sistema para gestionar excursiones y reservas. El sistema debe permitir registrar excursiones de diferentes tipos, calcular el precio final de cada excursión según sus características, y realizar reservas para los clientes interesados. Además, el sistema debe proporcionar estadísticas sobre las reservas realizadas y permitir la persistencia de datos en archivos para su posterior recuperación.

En este parcial, no les proporcionamos el diagrama UML. **Es su tarea analizar el problema y diseñar el UML de Clases.**

## **Capa Modelo**

#### **Clase Abstracta: Excursion (Implementa** Serializable**)**

## **Atributos:**

## int id: Identificador único que debe incrementarse automáticamente, comenzando en 1, esta será controlada por el **ExcursionRepository**.

## String titulo: Título descriptivo de la excursión con un mínimo de 3 caracteres.

## double precioBase: Precio base de la excursión.

## int cupoMaximo: Máximo número de reservas permitidas (No puede ser un número negativo).

## transient GestorReservas gestorReservas: Objeto encargado de gestionar las reservas asociadas a esta excursión, se instanciara en el constructor y se usará el generador de ruta para poder persistir las reservas de la excursión.

## List<String> actividades: Lista de actividades incluidas en la excursión.

## **Métodos:**

* boolean realizarReserva(String cliente): Realiza una reserva si hay cupos disponibles, la lógica debe de ser manejada por el gestorReservas.
* abstract double calcularPrecioFinal(): Calcula el precio final según el tipo de excursión.
* void agregarActividad(String actividad): Permite agregar actividades personalizadas.
* String mostrarResumen(): Devuelve un resumen de la excursión con actividades y cupos disponibles.
* private String generarFilePath(): Crea una ruta basada en el ID.

“reservas\_excursion\_{id}.dat"

* Optional<List<Reserva>> traerReservas(): Devuelve las reservas de una excursión.

#### **Clase ExcursionAventura (Subclase de Excursion)**

## **Atributos:**

## int nivelDificultad: Nivel de dificultad de la excursión (1 a 5).

## double seguroAdicional: Costo adicional opcional por seguro (puede variar según la dificultad).

## **Métodos:**

## calcularPrecioFinal(): El precio final de una excursión de aventura se calcula aplicando un incremento al precio base según el nivel de dificultad

## Por cada nivel de dificultad, se aplica un incremento del **10%** sobre el precio base.

## precioFinal = (precioBase \* (1 + 0.1 \* nivelDificultad)) + seguroAdicional.

* String consejosSeguridad(): Devuelve consejos de seguridad basados en el nivel de dificultad.

#### **Clase Reserva (Implementa** Serializable**)**

## **Atributos:**

## int idExcursion: Identificador de la excursión reservada.

## String cliente: Nombre del cliente que realiza la reserva.

## String fechaReserva: Fecha de la reserva (registrada automáticamente).

## double precio: Precio final de la reserva, lo puedes calcular en el metodo realizarReserva() de **Excursion.**

## boolean pagoConfirmado: Indica si el cliente ha confirmado el pago.

* **Métodos**:

## boolean confirmarPago(): Marca la reserva como pagada.

## String detalleReserva(): Devuelve los detalles completos de la reserva.

#### 

#### 

#### **Capa Negocio**

##### **Clase GestorExcursiones**

Gestiona las excursiones y utiliza el repositorio para manejar la persistencia.

* **Atributos:**
  + Repository<T> repository: Repositorio que almacena y gestiona las excursiones.
* **Constructor:**
  + Verifica si el archivo de excursiones existe. Si existe, carga los datos en el repositorio; en caso contrario, inicializa el repositorio con una lista vacía.
* **Métodos públicos:**
  + boolean agregarExcursion(Excursion excursion): Agrega una excursión al repositorio.
  + Optional<Excursion> buscarPorId(int id): Busca una excursión por su ID.
  + List<Excursion> filtrarExcursiones(Predicate<Excursion> criterio): Recibe un predicado y devuelve una lista de excursiones que cumplan el criterio.
  + List<Excursion> consultarExcursionesPorPrecio(double precioMin): Consulta excursiones cuyo precio final sea mayor o igual al valor proporcionado.

##### **Clase GestorReservas**

Gestiona las reservas y utiliza un repositorio para manejar la persistencia.

* **Atributos:**
  + Repository<T>repository: Repositorio que almacena y gestiona las reservas.
  + final int cupoMaximo: Máximo número de reservas permitido.
* **Constructor:**
  + Verifica si el archivo de reservas existe. Si existe, carga los datos en el repositorio; si no, inicializa el repositorio con una lista vacía.
* **Métodos públicos:**
  + boolean realizarReserva(Reserva reserva): Realiza una reserva para el cliente si hay cupo disponible y guarda los cambios en el repositorio.
  + List<Reserva> getReservas(): Devuelve la lista de todas las reservas.
  + double calcularIngresos(Predicate<Reserva> filtro): Usa un predicado para filtrar reservas y calcular ingresos específicos, como aquellos confirmados o en una fecha determinada.
  + double calcularIngresosTotales(): Calcula los ingresos totales por reservas confirmadas.

#### **Capa Persistencia**

##### **Interfaz Genérica: Repository<T>**

Define las operaciones del repositorio.

* **Métodos:**
  + void add(T entity): Agrega una entidad.
  + void remove(int id): Elimina una entidad por su ID.
  + Optional<T> findById(int id): Busca una entidad por su ID.
  + List<T> findBy(Predicate<T> criterio): Permite buscar entidades basándose en un criterio definido como lambda.
  + List<T> getAll(): Devuelve todas las entidades.

##### 

##### **Clase ExcursionRepository (Implementa Repository<Excursion>)**

Gestiona la lista de excursiones y su persistencia.

* **Atributos:**
  + List<Excursion> excursiones: Lista de excursiones.
  + GestorPersistencia<Excursion> persistencia: Maneja la serialización de datos.
  + String filePath: Ruta del archivo de persistencia.
  + int ultimoId: Última id registrada en el archivos de persistencia.
* **Métodos:**
  + Implementa los métodos CRUD definidos en la interfaz Repository<T>.

##### 

##### **Clase ReservaRepository (Implementa Repository<Reserva>)**

##### Gestiona la lista de reservas y su persistencia.

##### **Atributos:**

##### List<Reserva> reservas: Lista de reservas.

##### GestorPersistencia<Reserva> persistencia: Maneja la serialización de datos.

##### String filePath: Ruta del archivo de persistencia.

##### **Métodos:**

##### Implementa los métodos CRUD definidos en la interfaz Repository<T>.

##### 

##### **Interfaz Genérica: Serializer<T>**

Define métodos para serialización y deserialización de los archivos (.dat).

* **Métodos:**
  + <T> void guardarDatos(String filePath, List<T> data): Escribe datos en formato binario.
  + <T> List<T> cargarDatos(String filePath): Lee datos desde un archivo binario.

##### **Clase GestorPersistencia (Implementa Serializer<T>)**

* **Métodos:**
  + Implementa la escritura y lectura en formato binario para cualquier tipo de datos.

## 

## **Capa Test**

***Nota****: Al comienzo de cada prueba, incluir System.out.println("Test 1-1"); para indicar el caso en ejecución.*

### **Pruebas de Creación y Gestión de Excursiones**

Primero crea el **GestorExcursiones** con el que se guardaran las excursiones que intentes crear

#### **1.1 Crear y agregar una excursión de tipo ExcursionAventura con los siguientes atributos:**

* **titulo**: "Rafting en el Río Claro"
* **precioBase**: 1500.0
* **cupoMaximo**: 10
* **nivelDificultad**: 3

**Salida esperada**:

* "Excursión de tipo Aventura agregada exitosamente con ID 1."

#### **1.2 Intentar crear una excursión con un cupo negativo:**

* **titulo**: "Excursión en la Montaña"
* **precioBase**: 800.0
* **cupoMaximo**: -5

**Salida esperada**:

* Excepción lanzada: "El cupo máximo debe ser mayor que 0."

#### **1.3 Intentar crear una excursión sin un título:**

* **titulo**: ""
* **precioBase**: 1000.0
* **cupoMaximo**: 15

**Salida esperada**:

* Excepción lanzada: "El título de la excursión no puede estar vacío."

#### 

#### **1.4 Intentar crear una excursión con un título menor a 3 caracteres:**

* **titulo**: "Ra"
* **precioBase**: 800.0
* **cupoMaximo**: 10

**Salida esperada**:

* Excepción lanzada: "El título debe tener al menos 3 caracteres."

#### **1.5 Calcular el precio final de una excursión de aventura:**

* **Excursión**: "Rafting en el Río Claro"
* **precioBase**: 1500.0
* **nivelDificultad**: 3
* **seguroAdicional**: 200.0

**Salida esperada**:

* Precio Final: 2150.0

#### **1.6 Agregar actividades a una excursión de tipo ExcursionAventura:**

* **Excursión**: "Rafting en el Río Claro"
* **Actividades**: "Navegar el río", "Instrucción sobre seguridad"

**Salida esperada**:

* Actividades agregadas correctamente: ["Navegar el río", "Instrucción sobre seguridad"]

### **Pruebas de Reserva**

#### **2.1 Realizar una reserva para una excursión:**

* **Excursión**: "Rafting en el Río Claro" (ID 1)
* **Cliente**: "Carlos Pérez"

**Salida esperada**:

* "Reserva realizada con éxito para el cliente Carlos Pérez."

#### 

#### 

#### **2.2 Intentar realizar una reserva para una excursión sin cupos disponibles:**

* **Excursión**: "Rafting en el Río Claro" (ID 1) con cupo máximo 1 (ya realizado un reserva previa).
* **Cliente**: "Ana García"

**Salida esperada**:

* "No hay cupos disponibles para la excursión."

#### **2.3 Confirmar pago de una reserva:**

* **Reserva**: "Carlos Pérez" para "Rafting en el Río Claro"

**Salida esperada**:

* "Pago confirmado para la reserva de Carlos Pérez."

#### **2.4 Consultar el detalle de una reserva:**

* **Reserva**: "Carlos Pérez" para "Rafting en el Río Claro"

**Salida esperada**:

* Detalles completos de la reserva, incluyendo nombre del cliente, excursión, fecha, y precio final.

### **Pruebas de Métodos con Predicados**

#### **3.1 Filtrar excursiones por un precio mínimo:**

* **Criterio**: Precio mínimo de 1200.0
* **Excursiones**: "Rafting en el Río Claro" (precioBase: 1500.0), "Caminata en la Selva" (precioBase: 1200.0)

**Salida esperada**:

* Listado de excursiones que cumplen con el criterio de precio mayor o igual a 1200.0.

#### **3.2 Filtrar excursiones según un nivel de dificultad específico:**

* **Criterio**: Nivel de dificultad 3
* **Excursiones**: "Rafting en el Río Claro" (nivelDificultad: 3), "Caminata en la Selva" (nivelDificultad: 2)

**Salida esperada**:

* Listado de excursiones con nivel de dificultad igual a 3.

### **Pruebas de Cálculo de Ingresos**

#### **4.1 Calcular los ingresos totales por reservas confirmadas:**

* **Excursión**: "Rafting en el Río Claro"
* **Reserva confirmada**: "Carlos Pérez"

**Salida esperada**:

* "Ingresos totales: 1950.0" (precio final de la excursión)

#### **4.2 Calcular los ingresos filtrados por reservas confirmadas con un criterio específico (ejemplo: solo reservas confirmadas):**

* **Criterio**: Reserva confirmada
* **Excursión**: "Rafting en el Río Claro"
* **Reserva confirmada**: "Carlos Pérez"
* **Reserva no confirmada**: "Ana García"

**Salida esperada**:

* "Ingresos filtrados: 1950.0" (solo la reserva confirmada)

### **Objetivos de Aprobación No Directa (Calificación de 4 a 5 puntos):**

1. **Implementación Básica de Clases y Métodos:**El estudiante debe implementar las clases Excursion, ExcursionAventura, y Reserva, con los atributos y métodos solicitados. La clase Excursion debe ser abstracta, y las subclases deben implementar el método calcularPrecioFinal().
2. **Validaciones Fundamentales:**Se evaluará la implementación básica de validaciones, como impedir la creación de excursiones con títulos vacíos o cupos negativos. Sin embargo, pueden faltar algunas validaciones avanzadas (e.g., títulos con menos de 3 caracteres).
3. **Persistencia Inicial de Datos:**El sistema debe permitir guardar y cargar excursiones y reservas utilizando la clase PersistenciaDatos, aunque pueden existir errores menores al recuperar o guardar los datos.
4. **Gestión de Reservas Limitada:**Se espera que el sistema permita realizar reservas, pero podrían presentarse errores al gestionar duplicados, límites de cupos, o excursiones inexistentes.

### **Objetivos de Aprobación Directa (Calificación de 6 a 10 puntos):**

1. **Persistencia de Datos Sólida:**La persistencia debe ser totalmente funcional, permitiendo guardar y cargar excursiones y reservas sin errores, incluyendo casos de archivos inexistentes.
2. **Entrega de UML:**

El alumno debe entregar un diagrama UML que describa las clases principales del sistema, sus relaciones y métodos.

1. **Pruebas Exhaustivas:**Las pruebas deben cubrir todos los casos del enunciado, verificando la creación de excursiones, realización de reservas, manejo de excepciones, y consultas.
2. **Consistencia en la Lógica del Sistema:**

El alumno debe asegurar que la lógica del sistema sea consistente y que las listas de servicios se actualicen y consulten correctamente según las especificaciones del parcial, demostrando una comprensión profunda de la interacción entre las distintas clases y métodos.

1. **Buenas Prácticas de Programación:**El código debe seguir principios **DRY** y **SOLID**, estar bien estructurado y documentado, facilitando su comprensión y mantenimiento.